Title	Approximate GCD of Multivariate Polynomials (Theory and Application in Computer Algebra)
Author(s)	Zhi, LiHong; Noda, Matu-Tarow
Citation	数理解析研究所講究録 (2000), 1138: 64-76
Issue Date	2000-04
URL	http://hdl.handle.net/2433/63825
Right	
Type	Departmental Bulletin Paper
Textversion	publisher

数理解析研究所講究録
1138巻 2000年 64-76

Approximate GCD of Multivariate Polynomials

Dept. of Computer Science, Ehime University LiHong Zhi *
Dept. of Computer Science, Ehime University M.-T. Noda †

1 Introduction

Problem. Given two multivariate polynomials with inexact coefficients, how to produce a 'satisfactory' GCD for a given error tolerance ϵ?

$$\begin{aligned} F &= C(x_1,\ldots,x_n)\widetilde{F}(x_1,\ldots,x_n) + \mathsf{O}(\epsilon(x_1,\ldots,x_n)) \\ G &= C(x_1,\ldots,x_n)\widetilde{G}(x_1,\ldots,x_n) + \mathsf{O}(\epsilon(x_1,\ldots,x_n)) \end{aligned}$$

Example 1.

$$\begin{aligned} F &= 0.25 - 0.25x^2 - 0.25y^2 - 0.9999xy + xy^3 + yx^3 \\ &= (x^2+y^2-1)(xy-0.25) - 10^{-5}xy, \\ G &= -0.00001 + y - 1.00001x + xy^2 - yx^2 + x^3 - y^3 \\ &= (x^2+y^2-1)(x-y) - 10^{-5}(x+1) \end{aligned}$$

For $\epsilon = 10^{-5}$, the coefficients are rounded in the obvious way, the GCD is

$$x^2 + y^2 - 1.$$

The following three results corresponding to the three methods that we will introduce. (Use Maple, Digits=10).

1. Subresultant method:
$$1.000010000 - 1.y^2 - 1.x^2.$$
2. Modular method:
$$-0.9998612141 + 0.999795936y^2 + 1.000000000x^2.$$
3. Hensel lifting method:
$$-1.000009760 + 1.000000000x^2 + .9999982326y^2.$$

*lzhi@hpc.cs.ehime-u.ac.jp
†noda@hpc.cs.ehime-u.ac.jp

2 Subresultant PRS

The Subresultant PRS algorithm is the Euclidean algorithm using pseudo-division in a polynomial ring.

$$\{P_1 = F, P_2 = G, \ldots, P_k \neq 0, P_{k+1} = 0.\}$$

Where

$$\beta_i P_{i+1} = \text{remainder}(\alpha_i P_{i-1}, P_i), i = 2, 3, \ldots .$$

here,

$$\begin{aligned} \alpha_i &= \text{lc}(P_i)^{d_i+1}, \quad d_i = \deg(P_{i-1}) - \deg(P_i); \\ \beta_2 &= 1, \quad \gamma_2 = 1; \\ \beta_i &= \text{lc}(P_{i-1}), \quad \gamma_i = \text{lc}(P_{i-1})^{d_i-1} \gamma_{i-1}^{1-d_{i-1}}, i \geq 3. \end{aligned}$$

$\text{lc}(P)$ denotes the leading coefficient of P. The above method has been extended to polynomials with floating-point coefficients by Prof.Noda and Sasaki [3].

Noda and Sasaki's Approx-GCD.

1. Normalization of remainders in PRS. Let R and Q be pseudo-remainder and pseudo-quotient of polynomials F and G, i.e.,

$$R = \text{lc}(G)^{d+1} F - QG, \quad d = \deg(F) - \deg(G).$$

The normalized pseudo-remainder $\widetilde{R}$ is defined as

$$\widetilde{R} = \frac{R}{\max(\| \text{lc}(G)^{d+1} \|, \| Q \|)}.$$

2. Rounding of P_k. Normalize P_k as $\| P_k \| = 1$, round off the coefficients of P_k at $O(\epsilon)$.

3. Return the primitive part of P_k.

There are several points we should be careful with the floating point computation of Subresultant sequence.

1. For the exact coefficients, the Subresultant PRS is a compromise between the Primitive ($\beta_i = \text{cont}(\text{prem}(P_{i-1}, P_i))$) and Euclidean($\beta_i = 1$). At each step, the pseudo-remainder is reduced by a factor β_i that can be computed much more efficiently than the content. For the inexact coefficients, β_i is only an approximate factor of $\text{prem}(P_{i-1}, P_i)$. How to compute β_i and divide it out is still under investigation.

2. Despite the optimizations in the Subresultant algorithm, the growth of degrees of the coefficients is exponential w.r.t. the number of variables. But the GCD is of small degree w.r.t. each variable. See the Example 2.

Example 2.

$$\begin{aligned} F &= (x + 2y + 1)(y^4 + xy + 2), \\ G &= (x + 2y + 1)(x^3 + 3(x + 1)y^2 + 1.00001). \end{aligned}$$

$P_5 = O(10^{-5})$, and

$$\begin{aligned} P_4 = {} & 0.005847950905x^9 - 0.01754385272x^8 + 0.03508770544x^7 - \\ & 0.03508760310x^6 + 0.3859645844x^5 - 0.1929825554x^4 + \\ & 0.1754382056x^3 + 0.4035123846x^2 + 1.666665834x + \\ & 2.111112222 + (0.01169590181x^8 - 0.04678360725x^7 - \\ & 0.1169590181x^6 - .1871342536x^5 + .9590637732x^4 + \\ & 1.345029718x^3 + 1.695908307x^2 - .8888867398x + \\ & 4.222223888)y. \end{aligned}$$

After normalization of P_4 and dividing out the content, we get the approximate GCD:

$$2.000000000y + 1.000000001 + 1.000000000x.$$

3. The round off at $O(\epsilon)$ can produce unsatisfactory answers. (Similar problem as using Noda's approximate GCD for univariate polynomials.)

   Example 3.

$$\begin{aligned} F &= y^4 + y + x + 1, \\ G &= y^3 - \eta y + x^2. \end{aligned}$$

   Let $\eta = 10^{-3}$ and $\epsilon = 10^{-6}$, by the Subresultant method, we get $P_5 = O(\epsilon)$ and

$$P_4 = (-0.995974x + 0.996977 - 1.00201x^2 + x^3)y - 10.x^2 + 0.997984 + 0.002012x.$$

   But the remainders of F, G w.r.t. P_4 are $O(1)$.

4. Subresultant method is very inefficient for finding out that polynomials F and G have no approximate GCD. We have to continue the computation until the last step.

   Example 4.

$$\begin{aligned} F &= (1 + x + y_1 + y_2 + y_3 + y_4 + y_5 + y_6)(2 + x + y_1 + y_2 + y_3 + y_4 + y_5 + y_6), \\ G &= (1 + x^2 + y_1^2 + y_2^2 + y_3^2 + y_4^2 + y_5^2 + y_6^2)(-3y_1x^2 + y_1^2 - 1). \end{aligned}$$

   The last one in the Subresultant PRS is just the resultant of F and G w.r.t. x. We can check that it is nonzero and has 3591 terms with total degree 10 in $y_1, y_2, ..., y_6$.

3 Modular Algorithm

Multivariate polynomial GCD problems are reduced to univariate problems by using(repeatedly) evaluation homomorphisms to eliminate variables, and reconstruct the GCD of the original inputs from these "images" using Chinese Remainder Algorithms.

$$\begin{array}{lcll} \mathbb{R}[x_1,\ldots,x_n] & \times & \mathbb{R}[x_1,\ldots,x_n] & \overset{\text{GCD}}{\rightarrow} \quad \mathbb{R}[x_1,\ldots,x_n] \\ \\ \mod (x_n - a_n) & \downarrow & & \downarrow \mod (x_n - a_n) \\ \\ \mathbb{R}[x_1,\ldots,x_{n-1}] & \times & \mathbb{R}[x_1,\ldots x_{n-1}] & \overset{\text{GCD}}{\rightarrow} \quad \mathbb{R}[x_1,\ldots,x_{n-1}] \end{array}$$

Choose a main variable, suppose x_n, determine the content and primitive part of F and G considered as multivariate polynomials in $\mathbb{R}[x_1,\ldots,x_{n-1}]$ with coefficients from $\mathbb{R}[x_n]$, then $\text{GCD}(F,G) = \text{UGCD}(\text{cont}(F),\text{cont}(G))\cdot\text{GCD}(\text{pp}(F),\text{pp}(G))$. (UGCD computes univariate GCD). So in the following, we suppose F and G are primitive polynomials, and $\text{lc}(P)$ denotes the leading coefficient of the polynomial P in $\mathbb{R}[x_1,\ldots,x_{n-1}]$.

3.1 Dense Modular Algorithm

1. Bound the number of homomorphisms.
   $B = \min(\deg_{x_n}(F), \deg_{x_n}(G)) + \deg_{x_n}(c)$,
   where $c(x_n) = \text{UGCD}(\text{lc}(F), \text{lc}(G))$.

2. Choose lucky evaluation points b's.

   (a) Compute $C_b = \text{GCD}(F_b, G_b)$, where
   $F_b \equiv F \mod (x_n - b), G_b \equiv G \mod (x_n - b)$.

   (b) Normalize C_b so that $\text{lc}(C_b) = c(b)$, i.e., $C_b = c(b)\cdot\text{lc}(C_b)^{-1}\cdot C_b$.

   (c) Interpolate the GCD C from C_b's.

3. Return $\text{pp}(C)$.

Let us see the advantage and disadvantage of the algorithm.

1. The algorithm can uncover the small GCD very fast. Especially, in the case GCD = 1, we only need a few times lucky homomorphisms to discover it. For the Example 4, choose $y_1 = 2, y_2 = y_3 = \cdots = y_6 = 0$.

   $$\begin{aligned} F &\longrightarrow (3+x)(4+x), \\ G &\longrightarrow (5+x^2)(-6x^2+3). \end{aligned}$$

   It is easy to see that GCD of the two univariate polynomial is 1. Since the homomorphic images of of F and G do not decrease in degree w.r.t. x, we know F and G should be relatively prime.

2. But a typical implementation of this algorithm is prepared to compute many modular homomorphisms if a non-trivial GCD must be constructed, i.e., the number of homomorphisms needed is exponential in the number of variables.

Example 5

$$\begin{aligned} F &= (1+x^5+y_1^5+y_2^5+y_3^5+y_4^5)(-2+x^5+y_1^5+y_2^5+y_3^5+y_4^5), \\ G &= (1+x^5+y_1^5+y_2^5+y_3^5+y_4^5)(2+x^5+y_1^5+y_2^5+y_3^5+y_4^5). \end{aligned}$$

Since the GCD of degree 5 w.r.t each variable, it takes about $6^4 = 1296$ evaluations to find the GCD.

3. The normalization of the leading coefficient of C_b as $c = \mathrm{UGCD}(\mathrm{lc}(F), \mathrm{lc}(G))$ makes all coefficients of GCD C are polynomials in $x_2, \ldots, x_n$ since

$$\mathrm{lc}(\mathrm{GCD}(F,G))|\mathrm{UGCD}(\mathrm{lc}(F), \mathrm{lc}(G)).$$

So we can interpolate the coefficients individually and then divide out the content. Especially, if $\deg_{x_n}(c) = 0$, we know $\mathrm{GCD}(F,G)$ should be monic. On the other hand, if $\deg_{x_n}(c) > 0$, and $\mathrm{lc}(\mathrm{GCD}(F,G))$ is a non-trivial factor of c, then we interpolate a higher polynomial(with leading coefficient c) and divide out the non-trivial cofactor. Some computation is wasted.

Example 6

$$\begin{aligned} F &= (xy_1y_2y_3y_4y_5 - 1)(xy_1y_2y_3y_4y_5 + 3), \\ G &= (xy_1y_2y_3y_4y_5 - 1)(xy_1y_2y_3y_4y_5 - 3). \end{aligned}$$

Since $c = \mathrm{UGCD}(\mathrm{lc}(F), \mathrm{lc}(G)) = x^2$, perform the first two steps in the algorithm, we will get

$$C = x^2y_1y_2y_3y_4y_5 - x, \quad \text{and}$$

$$\mathrm{GCD}(F,G) = \mathrm{pp}(C) = C/x = xy_1y_2y_3y_4y_5 - 1.$$

4. The algorithm can be extended to polynomials with inexact coefficients with very little modifications.

3.2 Sparse Modular Algorithm.

A "skeleton"(degree and sparsity) of the GCD in the variables is computed by random evaluation homomorphisms. The sparse interpolation ignores the zero coefficients from the "skeleton".

Corless's Approx-GCD(bivariate case)[1].

1. Choose random evaluation α, β_i of x and y to compute T_x nonzero terms in $\mathrm{GCD}(F(x,\beta_i), G(x,\beta_i))$ and T_y nonzero terms in $\mathrm{GCD}(F(\alpha,y), G(\alpha,y))$.

2. Solve M monic $\mathrm{GCD}(F(x,\beta_i), G(x,\beta_i))$ for random choose β_i, where $M \geq T_yT_x/(T_x - 1)$.

3. Interpolate all coefficients simultaneously.

For the last step, it means to find the zero eigenvector of the following matrix:

$$\begin{bmatrix} B & & & Y_1B \\ & B & & Y_2B \\ & & \ddots & \vdots \\ & & B & Y_pB \end{bmatrix}$$

where B is the M by T_x Vandermonde matrix associated with β_i raised to the known powers, and Y_i is the diagonal matrix of the known values of the $i-th$ coefficient at each β_i. Let

$$G_i = (B^TB)^{-1}B^TY_iB,$$

and

$$M = \sum_{i=1}^{T_y-1} (Y_iB)^T(Y_iB) - (Y_iB)^TBG_i.$$

Then the eigenvector of matrix M gives the coefficients of the denominator polynomial and the coefficients of the numerator polynomials is then simply a matter of matrix multiplication. For the detail, see Corless's paper in ISSAC95.

Corless's method can be extended to multivariate cases. For Example 5, after some random evaluations, we get the estimate of nonzero terms in all variables.

$$T_x = T_{y_1} = T_{y_2} = T_{y_3} = T_{y_4} = 2.$$

So it needs about $4^4 = 256$ random evaluations. The matrix B is 4×2. Compared with the dense modular method(1296), it needs much less number of homomorphisms. We get the following GCD by Corless's algorithm:

$$0.9999999706 + 0.9999999998x^5 + 0.9999977716y_1^5 + 1.000000018y_2^5 + 0.9999999701y_3^5 + 1.000000179y_4^5.$$

Note that in the second step of Corless's method, all GCD are computed as monic, whereas the true GCD may have a leading coefficient in all variables. Hence the coefficients of the computed GCD's will be rational functions. For the floating point coefficients, all the coefficients have to be interpolated at the same time to get the same denominator. As in the dense interpolation, normalize the GCD to have the leading coefficient $\mathrm{GCD}(\mathrm{lc}(F), \mathrm{lc}(G))$, all coefficients will be polynomials not rational functions. So we can interpolate the coefficients independently. These two different techniques all have advantage and disadvantage, the efficiency and stabability of these techniques should be checked.

For the Example 5,

$$\mathrm{GCD}(\mathrm{lc}(F), \mathrm{lc}(G)) = \mathrm{GCD}(1, 1) = 1,$$

$\mathrm{GCD}(F, G)$ is monic. So the denominator is 1 and we only need $2^4 = 16$ times evaluations to interpolate all coefficients independently. The following GCD is computed by modified sparse modular method:

$$0.9999999996 + 1.000000000x^5 + 0.9999999894y_1^5 + 0.9999999944y_2^5 + 1.000000001y_3^5 + 1.00000016y_4^5.$$

For Example 6, the leading coefficient of the GCD is not monic. Both methods takes almost same time to get the similar results.

By the dense modular method, we get

$$-0.9999995801 + .9999999990xy_1y_2y_3y_4y_5.$$

The Corless's method gives:

$$1.000001006 - 1.000000000xy_1y_2y_3y_4y_5.$$

But if we increase the number of variables in Example 6 to 7, both methods get the results after one hour.

Example 6'

$$\begin{aligned} F &= (-1 + xy_1y_2y_3y_4y_5y_6)(3 + xy_1y_2y_3y_4y_4y_5y_6), \\ G &= (-1 + xy_1y_2y_3y_4y_5y_6)(-3 + xy_1y_2y_3y_4y_5y_6). \end{aligned}$$

$$-1.000022550 + xy_1y_2y_3y_4y_5y_6. \text{ (dense modular method)}$$

$$-1.000011342 + 1.000000000xy_1y_2y_3y_4y_5y_6. \text{ (Corless's method)}$$

With Subresultant method for this example, only one time pseudo-remainder will be enough to find the GCD

$$-1. + 1.xy_1y_2y_3y_4y_5y_6.$$

It takes less than one second.

3.3 Problems in Modular Algorithm.

There are several problems related with modular method.

1. The sparse modular algorithm is fast but also probabilistic. The algorithm is not efficient for dense GCDs and could also be unlucky.

   Example 7.

$$\begin{aligned} F &= (1 + x + y_1 + y_2 + y_3 + y_4 + y_5)^2 \\ &\quad (-2 + x - (y_1 + y_2 + y_3 + y_4 + y_5)^2), \\ G &= (1 + x + y_1 + y_2 + y_3 + y_4 + y_5)^2 \\ &\quad (2 + x + y_1 + y_2 + y_3 + y_4 + y_5)^2. \end{aligned}$$

   Compute

$$T_x = T_{y_1} = T_{y_2} = T_{y_3} = T_{y_4} = T_{y_5} = 3.$$

The GCD is monic and dense, we need about $3^5 = 243$ evaluations.

For Digits=10, we get the following GCD:

$$2.000287759x + 0.9999999990x^2 + 1.999309659y_1x +$$
$$1.999086523y_1 + 1.997897512y_2 + 1.997731189y_3 +$$
$$1.994760705y_4 + 1.995431666y_5 + 1.001337634y_1^2 +$$
$$1.001877601y_2^2 + 1.002023584y_3^2 + 1.005125000y_4^2 +$$
$$1.00550309y_5^2 - 0.112237449y_2y_4y_5 - 0.068170607y_2y_3y_5 -$$
$$0.120297930y_3y_4y_5 - 0.052576453y_1y_4y_5 - 0.038704547y_1y_2y_5 -$$
$$0.026331383y_1y_3y_5 + 0.083544389y_5^2y_2y_3 + 0.019901150xy_4y_5 +$$
$$0.064061577y_5^2y_1y_4 + 0.135066327y_5^2y_2y_4 + 0.147662364y_5^2y_3y_4 +$$
$$0.046139188y_5^2y_1y_2 + 0.035355942y_5^2y_1y_3 - 0.023787436y_5^2xy_4 +$$
$$0.019768305y_3^2y_5^2 + 0.046660040y_4^2y_5^2 + 0.012565150y_1^2y_5^2 +$$
$$0.014106929y_2^2y_5^2 - 0.04807766y_4y_5^2 - 0.010227795y_5y_1^2 -$$
$$0.012570726y_5y_2^2 - 0.015843060y_5y_3^2 - 0.038765561y_5y_4^2 -$$
$$0.01813891y_2y_5^2 - 0.02188706y_3y_5^2 + 2.039872287y_4y_5 +$$
$$1.997372839xy_4 + 1.997857982xy_5 + 2.006807072y_1y_4 +$$
$$2.006832635y_1y_5 + 2.014768902y_2y_4 + 2.015310152y_2y_5 +$$
$$2.015265793y_3y_4 + 2.017650837y_3y_5 + 1.000605701 +$$
$$1.998890923xy_2 + 1.998937297xy_3 + 2.005452365y_1y_2 +$$
$$2.003288032y_1y_3 + 2.009158854y_2y_3.$$

Some terms such as $y_2y_4y_5, \ldots, y_2y_3y_5^2$ which should be zeros but have small coefficients here. In the case Digits=20, we can get a better result:

$$1.99999999940488512x + 0.99999999891510172 + 1.00000000000000000x^2 +$$
$$2.00000000288431181y_1x + 2.00000000579956139y_1 + 2.00000000256551972y_2 +$$
$$2.00000000313751995y_3 + 2.00000000329417117y_4 + 2.00000000237025470y_5 +$$
$$0.99999999324365994y_1^2 + 0.99999999856962890y_2^2 + 0.99999999783472164y_3^2 +$$
$$0.99999999764400842y_4^2 + 0.99999999870709785y_5^2 + 1.99999999280280600y_4y_5 +$$
$$2.00000000179494181xy_4 + 2.00000000130023750xy_5 + 1.99999998240374590y_1y_4 +$$
$$1.99999998732962140y_1y_5 + 1.99999999224108744y_2y_4 + 1.99999999439435170y_2y_5 +$$
$$1.99999999056934526y_3y_4 + 1.99999999314482020y_3y_5 + 2.00000000159086091xy_2 +$$
$$2.00000000166604739xy_3 + 1.99999998565851368y_1y_2 + 1.99999998298935520y_1y_3 +$$
$$1.99999999245837234y_2y_3.$$

2. It is very important to choose good random evaluations due to the ill conditioned of the Vandermonde matrix for interpolation of polynomials. A variety of bounds for

the condition number of a Vandermonde matrix have been derived by Gautschi [2]. Let $V_n = V(\alpha_0, \alpha_1, \ldots, \alpha_{n-1}) \in \mathbb{R}^{n\times n}$, u_i be Chebyshev nodes in $[-1, 1]$.

Bounds and Estimates for $k_\infty(V_n)$

α_i	$k_\infty(V_n)$	$n=5$	$n=6$	$n=7$
equispaced $[0,1]$	$(4\pi)^{-1}\sqrt{2}\,8^n$	3687.7	29501.5	236012.5
equispaced $[-1,1]$	$\pi^{-1}\epsilon^{-\pi/4}(3.1)^n$	41.5	128.8	399.3
Chebyshev nodes $[-1,1]$	$\frac{3^{3/4}}{4}(1+\sqrt{2})^n$	46.7	112.8	272.4
$\alpha+\beta u_i, \alpha, \beta \in (1/2, 1)$	?	$10^3 \sim 10^4$	$10^4 \sim 10^5$	$10^5 \sim 10^6$

In Corless's paper, they suggested to randomly scale each variable and choose the random points as the last row in the table: $\alpha + \beta u_i$. It is still possible to get very large condition numbers in the case n is big. We should monitor the choice of random points and throw away the points with large condition numbers. Since the equispaced points and Chebyshev nodes are of low condition numbers, we start them as random evaluation points. It works in most cases.

" We enthusiastically endorse interpolations with 3 or 4 points, we are perhaps tolerant of 5 or 6; but we rarely go higher than that unless there is quite rigorous monitoring of estimated errors." (cited from " Numerical Recipes: The Art of Scientific Computing"[5].)

That means it is relatively dangerous to interpolate the non-monic GCD with degree larger than 3 and dense in some variables.

4 Hensel Lifting Algorithm.

The two input polynomials are reduced to two univariate polynomials whose GCD is then lifted back to the multivariate domain using a generalized Newton's iteration.

$$\begin{array}{ccccc} \mathbb{R}[x_1,\ldots,x_n] & \times & \mathbb{R}[x_1,\ldots,x_n] & \overset{\text{GCD}}{\to} & \mathbb{R}[x_1,\ldots,x_n] \\ \text{mod } I & \downarrow & & & \downarrow \text{ mod } I \\ \mathbb{R}[x_1] & \times & \mathbb{R}[x_1] & \overset{\text{GCD}}{\to} & \mathbb{R}[x_1] \end{array}$$

Where $I = (x_2 - a_2, \ldots, x_n - a_n)$.

The EZ-GCD Algorithm.

1. Choose a main variable, suppose x_1, find lucky evaluation homomorphism $I = (x_2 - a_2, \ldots, x_n - a_n)$.

2. $F_I = F \mod I, G_I = G \mod I$. Compute $C_I = \text{GCD}(F_I, G_I)$ and cofactors $\widetilde{F_I}, \widetilde{G_I}$.

3. If there is one of the cofactors which is prime to C_I, then use multivariate Hensel construction to lift C_I and the cofactor. Otherwise perform a square-free decomposition of either F or G.

Suppose P is a polynomial in $x_1, \ldots, x_n$ with leading coefficient p_m. Let $a_2, \ldots, a_n$ satisfy $p_m(a_2, \ldots, a_n) \neq 0$, $I = (x_2 - a_2, \ldots, x_n - a_n)$, $x = x_1, \mathsf{u} = \{x_2, \ldots, x_n\}$, and $G^{(0)}(x)$ and $H^{(0)}(x)$ be relatively prime polynomials satisfying

$$P(x, \mathsf{u}) = G^{(0)}(x)H^{(0)}(x).$$

The multivariate Hensel construction is to calculate polynomials $G^{(k)}(x, \mathsf{u})$ and $H^{(k)}(x, \mathsf{u})$ satisfying

$$P(x, \mathsf{u}) \equiv G^{(k)}(x, \mathsf{u})H^{(k)}(x, \mathsf{u}) \mod I^{k+1}.$$

There are three main steps in the multivariate Hensel construction.

1. Compute univariate polynomials $A_i(x)$ and $B_i(x)$ (i=0, 1, ... , m) which satisfying

$$\begin{cases} A_i(x)G^{(0)}(x) + B_i(x)H^{(0)}(x) = x^i, \\ \deg(A_i) < \deg(H^{(0)}), \quad \deg(B_i) \leq \deg(G^{(0)}). \end{cases}$$

2. From $G^{(i)}$ and $H^{(i)}$, $i < k$, we calculate

$$\Delta P^{(k)} \equiv P - G^{(k-1)}H^{(k-1)} \equiv \sum_{i=0}^{m} \Delta p_i^{(k)} x^i \mod I^{k+1}.$$

3. We construct $G^{(k)}$ and $H^{(k)}$ as

$$\begin{cases} G^{(k)} &= G^{(k-1)} + \sum_{i=0}^{m} \Delta p_i^{(k)} B_i \\ H^{(k)} &= H^{(k-1)} + \sum_{i=0}^{m} \Delta p_i^{(k)} A_i. \end{cases}$$

One method to compute A_i and B_i is by extended Euclidean algorithm. Another method is to form the Sylvester matrix M of the polynomials $G^{(0)}(x)$ and $H^{(0)}(x)$, then compute the inverse of M. The coefficients of A_i and B_i are the subvectors of M^{-1}. We adopt the second way since there should be more techniques to stabilize the computation of the inverse of the Sylvester matrix. It is clear that if $G^{(0)}(x)$ and $H^{(0)}(x)$ have a close common root, the matrix M will be very ill-conditioned and the multivariate Hensel construction will be unstable [6]. This may be caused by a bad evaluation or both F and G are not squarefree.

Example 8.

$$\begin{aligned} F &= (x^2 + y^3 + 1 + 10^{-3})(x^2 + xy + y^2 + 1), \\ G &= (x^2 + y^3 + 1 + 10^{-3})(x^2 + xy + 1). \end{aligned}$$

Choose the random value of y near 0, for example $y = 3/20$, then

$$\begin{aligned} C_I &= 1.00438 - 0.000001x + .999993x^2, \\ F_I &= 1.02251 + 0.15000x + 1.00000x^2, \\ G_I &= 1.00001 + 0.15000x + 1.00000x^2. \end{aligned}$$

C_I and F_I, G_I have very close roots. Apply the Hensel lifting to the C_I and G_I, we get GCD

$$C = 1.00439 - 0.000001x + 1.x^2 + 0.06742y + 0.450553y^2 + 0.996396y^3.$$

Check the remainders of F and G w.r.t C, we find they are $O(1)$.

As in the modular method, it is also important to choose good evaluation points. In most case, we only need to choose one or two values for each variable, resulting in much faster execution over modular method. If I is chosen so that as many a_i as possible are zero, then the algorithm preserve sparsity and can cope with sparse polynomials quite well. Otherwise it has the same exponential performance as in modular algorithm.

Example 5'.

$$\begin{aligned} F &= (1 + x^5 + y_1^5 + y_2^5 + y_3^5 + y_4^5)(-2 + x^5 + y_1^5 + y_2^5 + y_3^5 + y_4^5) \\ &\quad +10^{-5}(x + y_1 + y_2 + y_3 + y_4), \\ G &= (1 + x^5 + y_1^5 + y_2^5 + y_3^5 + y_4^5)(2 + x^5 + y_1^5 + y_2^5 + y_3^5 + y_4^5) \\ &\quad -10^{-5}(x + y_1^2 + y_2^2 + y_3^2 + y_4^2). \end{aligned}$$

Choose $y_1 = y_2 = y_3 = y_4 = 0$, we get two well separated univariate factors of G

$$\begin{aligned} C_I &= 0.9999953335 + 0.9999996664x^5 \\ G_I &= 2.000001667 + 1.000001666x^5 \end{aligned}$$

Performing the Hensel lifting, we get the GCD

$$0.9999956675 + 1.000000000x^5 + 1.000005997y_2^5 + 1.000005997y_3^5 + 1.000005997y_4^5 + 1.000005997y_1^5.$$

Example 7'.

$$\begin{aligned} F &= (1 + x + y_1 + y_2 + y_3 + y_4 + y_5)^2(-2 + x - (y_1 + y_2 + y_3 + y_4 + y_5)^2) \\ &\quad -10^{-5}(x + 2y_1 + y_2 + 6y_3 + y_4 + y_5), \\ G &= (1 + x + y_1 + y_2 + y_3 + y_4 + y_5)^2(2 + x + (y_1 + y_2 + y_3 + y_4 + y_5))^2 \\ &\quad +10^{-5}(x + y_1^2 + y_2^2 + y_3^2 + y_4^2 + y_5). \end{aligned}$$

Choose $y_1 = y_2 = y_3 = y_4 = y_5 = 0$, we can find the approximate GCD very efficiently.

$$\begin{aligned} &1.999999913x + 1.99998451y_1x + 1.000000000x^2 + 1.99998557y_1 + \\ &1.99998557y_2 + 1.99998557y_3 + 1.99998557y_4 + 1.99998557y_5 + \\ &0.999948043y_1^2 + .999947983y_2^2 + 0.999948043y_3^2 + 0.999948043y_4^2 + \\ &0.999948043y_5^2 + 1.999896098y_4y_5 + 1.999896038y_2y_3 + 1.99989604y_2y_4 + \\ &1.99989604y_2y_5 + 1.99989610y_3y_5 + 1.999896098y_3y_4 + 0.9999993837 + \\ &1.99998451xy_2 + 1.99998451xy_5 + 1.99998451xy_3 + 1.99998451xy_4 + \\ &1.99989610y_1y_4 + 1.99989604y_1y_2 + 1.99989610y_1y_3 + 1.99989610y_1y_5. \end{aligned}$$

The normalization technique used in the modular algorithm can be used here to compute non-monic GCD, but this can produce very dense coefficients from sparse polynomials. For the Example 6 and 6', since the leading coefficients of F and G involve all variables, it is impossible to choose nonzero evaluation points for any variables. The evaluation at non-zero point makes polynomials become very dense. Much worse, the normalization of the leading coefficients makes the computation more difficult. Using Hensel lifting method for Example 6, we get the GCD:

$$-1.000000049 + xy_1y_2y_3y_4y_5.$$

But for Example 6', Hensel lifting fails.

5 Remarks

All the algorithms discussed above have been implemented in Maple. A large set of examples are tested. We observe the following facts.

1. The random evaluations should be selected with caution for sparsity and stability.

2. The leading coefficient problem should be treated carefully.

3. The Subresultant PRS algorithm is still useful when the polynomials are dense or with complicate leading coefficients.

As pointed by Corless's paper, we should perform the back error analysis to check if the answers are satisfying or not. It is not an easy task especially for multivariate polynomials. Formulate it as an optimization problem, there are too many variables if we do not consider the sparsity of the true GCD. The problem is still under investigation.

Since all methods are based on the approximate univariate GCD computation, we need to choose an efficient and stable algorithm for UGCD. Especially for Hensel lifting method, we also need the cofactor of the GCD. In our algorithm, we modify Noda's algorithm and use the result as an initial guess, then apply the optimization strategies introduced in [4] to improve it.

References

[1] Corless, R. M., Gianni, P. M., Trager, B. M. and Watt, S. M.: The singular value decomposition for polynomial systems, *Proc. ISSAC '95*, ACM Press, New York, 1995, 195–207.

[2] Gautschi, W.: How (un)stable are Vandermonde systems? *Lecture Notes in Pure and Applied Mathematics,* **124**(1990), 193-210.

[3] Noda, M.-T. and Sasaki, T.: Approximate GCD and its application to ill-conditioned algebraic equations, *J. Comput. Appl. Math.*, **38**(1991), 335–351.

[4] Chin, P., Corless, R. M. and Corliss, G. F.: Optimization strategies for approximate GCD problem, *Proc. ISSAC '98*, ACM Press, New York, 1998, 228-235.

[5] Press, W., Flannery, B., Teukolsky, S. and Vetterling, W.: Numerical Recipes: The Art of Scientific Computation, Cambridge U. Press, Cambridge, 1990.

[6] Sasaki, T. and Yamaguchi, S.: An analysis of cancelation error in multivariate Hensel construction with floating-point number arithmetic, *Proc. ISSAC '98*, ACM Press, New York, 1998, 1–8.